(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 638 187 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
25.07.2001 Bulletin 2001/30

(21) Application number: **93906168.5**

(22) Date of filing: **23.02.1993**

(51) Int Cl.[7]: **G06F 17/30**

(86) International application number:
PCT/US93/01590

(87) International publication number:
WO 94/19757 (01.09.1994 Gazette 1994/20)

(54) CATEGORIZING STRINGS IN CHARACTER RECOGNITION

KATEGORISIERENDE ZEICHENKETTEN IN DER ZEICHENERKENNUNG.

CATEGORISATION DES CHAINES DANS LA RECONNAISSANCE DES CARACTERES

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU MC NL PT SE

(43) Date of publication of application:
15.02.1995 Bulletin 1995/07

(73) Proprietor: **XEROX CORPORATION**
Rochester, New York 14644 (US)

(72) Inventors:
- **KAPLAN, Ronald, M.**
  Palo Alto, CA 94306 (US)
- **SHUCHATOWITZ, Robert**
  Brighton, MA 02135 (US)
- **MULLINS, Atty, T.**
  Boston, MA 02115 (US)

(74) Representative:
Walker, Antony James Alexander et al
W.P. Thomson & Co.,
Coopers Building,
Church Street
Liverpool L1 3AB (GB)

(56) References cited:
EP-A- 0 282 721
US-A- 4 034 343
US-A- 4 985 863
US-A- 5 003 614
WO-A-83/01329
US-A- 4 750 122
US-A- 4 989 258

- **REVUZ: "Dictionnaires et Lexiques Methodes et Algorithmes (LITP 91.44) ", 1991, AVAILABLE IN ISTITUTE BLAZE PASCAL , PARIS**
- **REVUZ: "", , UNIVERSITE DE PARIS,**

EP 0 638 187 B1

Description

[0001] The present invention relates to character recognition techniques that categorize strings, such as strings of characters or other elements.

[0002] Some conventional character recognition techniques improve recognition accuracy by taking into account different categories of character strings that can occur in text. The character recognition products of Xerox Imaging Systems, for example, employ several specialized recognition algorithms, each for a respective category of character string. One algorithm can look up words in a dictionary, another can recognize valid strings of arabic numerals with punctuation marks, and so forth. If a string of character candidates includes a candidate that has a substantial probability for more than one character category, the ambiguity can be resolved by applying each recognition algorithm to the possible string that would result from each probable character category. If one of the possible strings is recognized by one of the algorithms, the recognition result can be used to resolve the ambiguity. If two or more possible strings are recognized by different algorithms, the sequence of categories of strings can be taken into account to resolve the ambiguity.

[0003] US-A-4,003,022, describe string pattern recognition techniques. As shown and described in relation to Figs. 1-4 and 7, a character is decomposed into a symbolic string pattern that is fed to sequential logic. At column 3 lines 52-55 it is stated that the class for each symbol string pattern is determined by comparing it with all sequential logics and checking which has accepted the pattern.

[0004] Sinha, R.M.K. and Prasada, B., "Visual Text Recognition through Contextual Processing," *Pattern Recognition*, Vol. 21, No. 5, 1988, pp. 463-479, describe visual text recognition techniques. Dictionary methods are described beginning in the last paragraph on page 463. Section 2.1 on page 465 describes a partial dictionary. Section 2.6 on page 467 describes word boundary identification based on unambiguous punctuation marks. Section 2.7 on pages 467-468 describes heuristics use to identify word boundaries, match with the dictionary, and traverse through a modified Viterbi net. Section 3 on page 468 and Figs. 3 and 4 describe dictionary organization and search, and relate specifically to a trie structure based dictionary. Fig. 3 shows a node structure, including a NEXT field, an ALT field, a character, an end of word mark, and a flag. Page 473, right hand column, notes that errors are encountered due to several causes, including ambiguous punctuation marks within a word and mapping of numbers to a dictionary word.

[0005] Srihari, S.N., Hull, J.J., and Choudhari, R., "Integrating Diverse Knowledge Sources in Text Recognition," *ACM Transactions on Office Information Systems,* Vol. 1, No. 1, January 1983, pp. 68-87, describe text recognition techniques. Section 3 on pages 72-74 and Fig. 2 describe lexical organization, relating specifically to a letter trie. Fig. 2(a) shows the fields of a node, including: a token, CHAR; a word-length indicator array of bits, WL; an end-of-word tag bit, E; two pointers labeled NEXT and ALTERNATE.

[0006] WO-A-8301329 discloses items of data indicating a set of words, including stop word table 3010 and entry (candidate) words 3022, but nothing in this document suggests that these items of data include sequences of data units that begin with character data unit subsequences, have acceptance data and have ending sequences with category data units.

[0007] EP-A-0282721 describes dictionaries with paradigm numbers. The paradigm numbers provide information about part of speech, inflectional forms, and grammatical role. But this document does not teach or suggest sequences of data units that begin with character data unit subsequences, have acceptance data, and have ending subsequences with category data units.

[0008] Revuz D, Dictionaries and Lexicons, Methods and Algorithms a doctoral theses published 1991 and locatable of the Institute Blaze Pascal in Paris, France under the index LITP 91.44, describes a representation of an automaton in which a state includes an indicator of whether it is final or not and a pointer to a list of its outgoing transitions. As shown in Fig. 10, each transition includes a character, a pointer to the next transition in the list, and a pointer to the state to which the transition leads. Section 3.6a describes an automaton with multiple terminal states, with each of which is associated one of a set of 8400 DELAF codes, and section 1.2.2 indicates that the DELAF codes indicate the endings a simple word may take.

[0009] Revuz discloses that a DELAF code may be associated with a terminal state, but does not disclose a technique in which an ending subsequence for an acceptable string includes two category data units.

[0010] The present invention provides a product for providing information about strings of characters according to claim 1.

[0011] The present invention further provides a system according to claim 11 of the appended claims.

[0012] One aspect of the invention deals with basic problems in conventional character recognition techniques that take character string category into account.

[0013] As described above, some conventional techniques handle each character string category with a separate algorithm. This approach is time inefficient because each category's algorithm must be performed for each string of character candidates. Also, this approach is space inefficient because the data necessary to perform each category's algorithm, including instructions, must be stored separately.

[0014] This aspect is further based on the discovery of a technique that alleviates these basic problems. The technique integrates the algorithms for all categories of strings so that only one algorithm is performed for each string of character candidates. The technique also integrates the data for each category of string, so that the data used by the integrated algorithm is very compact.

[0015] This discovery is based on the observation that directed graph techniques, and specifically directed graphs that represent finite state machines, make it possible to integrate diverse string recognition algorithms. The acceptable categories of strings vary significantly, but each category can be expressed as a respective finite state machine (FSM). Even categories that include infinitely many possible strings, such as the arabic numbers, can be expressed as cyclic FSMs. Therefore, the same lookup algorithm could be used with a directed graph representing each category's FSM. To realize space efficiency, the FSMs of all the categories can be composed into a combined FSM, so that only one directed graph is necessary to represent the combined FSM. To realize time efficiency, the lookup algorithm is only executed once in recognizing a string of character candidates. To resolve a string that is acceptable as more than one of the categories, the directed graph includes data indicating one or more categories for each acceptable string.

[0016] String data according to the invention thus includes a sequence of data units that can be accessed using data indicating characters of a string. If the string is acceptable, the sequence of data units ends with a subsequence that includes information indicating that the string of characters is acceptable and information indicating a category set for the string. The processor of a character recognition system can therefore use the ending subsequence to obtain ending data indicating that the string is acceptable and indicating the string's category set.

[0017] The technique described above is especially advantageous for data structured so that both prefixes and suffixes of strings can be collapsed to eliminate redundancy: Because the category indicators are relatively few in number, the data would include many occurrences of each category indicator if there were a category indicator for each acceptable string. But if the category indicator of each string is stored at the end of the string's suffix, collapsing suffixes greatly reduces the number of category indicators in the data. Suffixes can be collapsed with well-known minimization algorithms applicable to directed graphs or other data representing FSMs.

[0018] String data that includes category indicators as described above can be stored by a data storage medium such as a floppy disk. The data storage medium can also include control data, such as instructions, that a processor can use in accessing the string data. The processor of a system performing character recognition can therefore use the control data and the string data to obtain data indicating whether a string of characters is an acceptable string and indicating a set of categories for the string if it is acceptable.

[0019] Embodiments of the invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Fig. 1 is a schematic block diagram showing components of a system in which a processor can use string data to obtain category set data for a string;
Fig. 2 is a flow chart showing general steps by which the processor in Fig. 1 can match a character using the string data;
Fig. 3 is a flow chart showing general steps by which the processor in Fig. 1 can obtain ending data using the string data;
Fig. 4 is a schematic diagram of a character subsequence showing information it includes and its links to other subsequences;
Fig. 5 is a flow chart showing steps in accessing subsequences like that in Fig. 4 in matching a character;
Fig. 6 is a flow chart showing steps in accessing a subsequence like that in Fig. 4 in obtaining ending data indicating whether a string is acceptable and its category set;
Fig. 7 is a schematic drawing showing a character subsequence of data units;
Fig. 8 is a flowchart showing more detailed steps for implementing the technique of Fig. 5;
Fig. 9 is a flowchart showing more detailed steps for implementing the technique of Fig. 6;
Fig. 10 is a schematic drawing of an N-bit vector for returning category set data;
Fig. 11 is a schematic block diagram showing components of a system using string data for recognition;
Fig. 12 is a schematic drawing of a software product including string data.

A. General Features

[0020] Figs. 1-2 illustrate general features of the invention. Fig. 1 illustrates general features of string data that can be accessed to obtain acceptance data and category set data. Fig. 2 illustrates general steps in using string data like that shown in Fig. 1.

[0021] Fig. 1 shows system 10, in which string data 12 can be used. String data 12 can, for example, be stored by a data storage medium accessible by processor 14; for example, string data 12 could be stored in a software product such as a floppy disk, CD-ROM, or the like. Processor 14 can also receive character data 16 indicating characters of

a string. Processor 14 can provide acceptance data 18 indicating that the string of characters is acceptable and also indicating a set of categories for the string.

[0022] Fig. 1 shows sequence of data units 30 within string data 12. If character data 16 indicates the characters "mix," for example, sequence 30 may include subsequence 32 indicating the character "m," subsequence 34 indicating the character "i," and subsequence 36 indicating the character "x." Each subsequence can include one or more data units.

[0023] A subsequence can include other information in addition to information indicating a character. For example, subsequence 36, in addition to information indicating the character "x," also includes information indicating that the string "mix" is acceptable and that its categories include word and roman numeral. This information can be used by processor 14 if it reaches the end of the string "mix" while accessing subsequence 36. Processor 14 can use this information to obtain acceptance data indicating that "mix" is an acceptable string and to obtain category set data indicating a set of categories for "mix" that includes word and roman numeral.

[0024] Fig. 2 shows steps processor 14 could follow in using string data 12 to match a character. The step in box 50 begins by receiving a request to match a character and an entry location. This entry location could be a location representing a start state of an FSM or it could be another location within string data 12 from which data units can be accessed that indicate a set of alternative characters that can follow the entry location. For example, it could be the location of a character's subsequence of data units. The data units that can be accessed from the entry location can include a respective subsequence of data units for each of the alternative following characters.

[0025] The step in box 52 accesses the entry location in string data 12. The step in box 60 then accesses the data units that indicate alternative following characters until it determines whether the character matches one of the characters that can follow the entry location. If so, the step in box 62 returns the location of the matching character's subsequence, but if the character does not match any of the alternative following characters, the step in box 64 returns data indicating that the string being matched is not acceptable.

[0026] Fig. 3 shows steps processor 14 could follow in using string data 16 to obtain ending data for a string. The step in box 70 begins by receiving a request for ending data and an ending location. The step in box 72 accesses the ending location in string data 12. The step in box 80 determines whether the string that ends at the ending location is an acceptable string. If so, the step in box 82 returns ending data indicating the string is acceptable and also indicating a set of categories for the string. If the string is not acceptable, the step in box 84 returns data indicating it is not acceptable. The steps in boxes 80, 82, and 84 can be performed using acceptance information and category set information from an ending subsequence, such as ending subsequence 36 in Fig. 1.

[0027] The steps in Figs. 2 and 3 can be used together in character recognition. For example, processor 14 could perform the steps in Fig. 2 for each character of a candidate string. If any character cannot be matched starting at the location at which the preceding character was matched, processor 14 could obtain data indicating the string is not acceptable. Upon matching the last character of the string, processor 14 could perform the steps in Fig. 3 at the location where the last character was matched to determine whether the string is acceptable and to obtain data indicating its category set if it is acceptable.

B. Implementation

[0028] The general features described above could be implemented in numerous ways on various machines. The invention has been implemented in the Medley version of the Lisp programming language available from Venue Corporation for execution on a Sun SparcStation and in the C programming language for execution on a Sun SparcStation, a Macintosh personal computer, and other machines.

1. Character Subsequences

[0029] Figs. 4-6 illustrate high level features of string data that can be used in implementing the invention. Fig. 4 shows a character subsequence and its relationship to other subsequences. Fig. 5 shows steps in accessing character subsequences in response to a request to match a character. Fig. 6 shows steps in accessing a character subsequence in response to a request for ending data.

[0030] Character subsequence 100 in Fig. 4 includes several types of information. Label information 102 indicates a character type. Next subsequence information 104 indicates whether subsequence 100 has a next subsequence and, if so, the location of the next subsequence. Alt subsequence information 106 indicates whether subsequence 100 has an alternative subsequence and, if so, the location of the alternative subsequence. Ending information 108 indicates whether a string that ends with a character matching label information 102 is an acceptable string and, if so, a category set for the string.

[0031] An interlinked set of character subsequences like that shown in Fig. 4 can be linked together to represent an FSM. As shown, character subsequence 100 could be the next or alternative subsequence for one or more other

subsequences.

[0032] Fig. 5 generally follows the steps in Fig. 2. The step in box 120 begins upon receiving a call to a function designated "Advance," which is called with a location and a character type. The step in box 122 uses the location to access a character subsequence. The step in box 122 also uses the subsequence's next information to obtain data indicating the location of the subsequence's next subsequence. The step in box 124 uses the data indicating the next subsequence's location to access it.

[0033] The step in box 126 begins an iterative loop that accesses the next subsequence and its alternative subsequences, if any, to find a match. The step in box 126 uses the current subsequence's label information to obtain data indicating a character type. The step in box 130 branches based on whether this character type is the same as the character type received in box 120. If so, the step in box 132 returns data indicating the current subsequence's location. But if not, the step in box 134 uses the current subsequence's alt information to obtain data indicating whether the subsequence has an alternative subsequence and, if so, the location of the alternative subsequence.

[0034] If the current subsequence does not have an alternative subsequence, the branch in box 140 leads to the step in box 142, which returns data indicating that the character received in box 120 cannot be matched. But if the subsequence has an alternative subsequence, the step in box 144 uses the data indicating the alternative subsequence's location to access it before returning to the step in box 126.

[0035] Fig. 6 generally follows the steps in Fig. 3. The step in box 170 begins upon receiving a call to a function designated "End Status," which is called with a location. The step in box 172 uses the location to access a character subsequence. The step in box 172 also uses the subsequence's ending information to obtain data indicating whether the subsequence is the ending subsequence of an acceptable string and, if so, a set of categories for the string.

[0036] If the subsequence is not the ending subsequence of an acceptable string, the branch in box 180 leads to the step in box 182, which returns data indicating that the string is not acceptable. But if the subsequence is the ending subsequence of an acceptable string, the step in box 184 returns data indicating the set of categories for the string, data which also implicitly indicates that the string is acceptable.

2. Encoding and Decoding

[0037] Character subsequences could be encoded and decoded in many ways in implementing the invention. Table I shows one way a byte value space can be allocated to permit use of some byte values both as character bytes and as category bytes. Fig. 7 shows a character subsequence in which each data unit is a byte value from the allocation of Table I. Fig. 8 shows how steps in Fig. 5 could be implemented for the allocation of Table I. Fig. 9 shows steps for obtaining category set data. Fig. 10 shows a bit vector for returning category set data.

[0038] The byte values in Table I range from 0 to 255, and hence are eight bits in length. N is the number of distinct six-bit labels, explained in more detail below. P_1 is the number of long pointer byte codes, and P_2 is the number of midsize pointer byte codes. As can be seen in Table I, the number of short pointer byte codes is $255 - (4N + P_1 + P_2)$.

[0039] As a result of the EOB byte, each character has only four different types, the character's four possible combinations of final, also referred to herein as acceptable, and alternative, also referred to herein as alt. Having only four types of characters is advantageous because the type of each character can be represented by two bits, which can be masked for comparison, avoiding the need to do a table lookup or other complicated decoding of a byte value. Therefore, rather than allocating blocks as in the FSM encoding application, Table I allocates four consecutive byte values to each of N - 2 characters.

TABLE I

Byte Values	Interpretation
0	EOB Byte
1 to $255 - (4N + P_1 + P_2)$	Short Pointer
$256 - (4N + P_1 + P_2)$ to $255 - (4N + P_1)$	Midsize Pointer
$256 - (4N + P_1)$ to $255 - 4N$	Long Pointer
$256 - 4N$ to $259 - 4N$	Tagmark or Category Types I-IV
$260 - 4N$ to 251	Characters or Categories Types I-IV
252-255	Table Label Types I-IV

[0040] Table I also allocates four consecutive byte values to a tagmark, which is not interpreted as a character but rather indicates that the preceding character is the end of an acceptable string. The bytes following a tagmark can thus

be interpreted as something other than characters, and Table I shows that the tagmark byte value and the character byte values can also be interpreted as category byte values. This interpretation is applied to a byte value in the tagmark and character/category ranges if it is accessed in a sequence of data units after accessing a tagmark.

[0041] Table I also allocates four consecutive byte values to a table label, which is interpreted neither as a character nor as a category, but rather indicates that the next bytes are a pointer to a table. Table labels are an optional technique that can be used to reduce the time necessary to search the string data representing an FSM in cases where a state in the FSM has a large number of outgoing transitions. Table labels have been found to be advantageous where an FSM transducer summarizes general rules, where an FSM permits compounding of acceptable strings, or where an FSM is used to classify strings that may be composites of acceptable strings and other elements such as punctuation marks. States in such situations resemble the FSM's start state, which has a large number of outgoing transitions.

[0042] Each table can be implemented with N - 1 entries, so that there is one entry for each of the label values, from the tagmark byte through the character/category bytes; because the table label has the highest label value, the table need not have an entry for the table label itself. The entries are of equal length, so that the entry for a given label value can be accessed by multiplying the six-bit label value by the entry length to obtain an offset into the table.

[0043] Each entry begins with a label byte. If a label value can be matched from the location preceding the table label, the label value's entry has a label byte with the same label value. But if the label value cannot be matched from the location preceding the table label, the label value's entry has a label byte with a different label value; for example, a table label byte could be used as the label byte for each entry that cannot be matched, since its label value is different than the label value of any of the entries.

[0044] Each entry also includes a pointer field following the label byte, and the pointer field is large enough to hold a full length pointer. In the current implementation, each full length pointer is three bytes, so that the entry length is four bytes. If a label value can be matched from the location preceding the table label, the pointer field of the label value's table entry contains a pointer to the next subsequence for the label or the special EOB byte if there is no next subsequence.

[0045] A table as described above can be shared by a state for which some of the table's entries are inappropriate. This can be done if the outgoing transitions for each state are ordered so that the table label is always the last transition. Preceding outgoing transitions can be used to provide information for transitions for which the table entry is inappropriate.

[0046] In general, the choice of which states should be followed by table labels involves a complex time-space tradeoff that may depend heavily on the specific data and task. The use of more tables improves search speed, but may increase the space needed to store the string data, which include the tables. In certain situations, such as states representing alternative terminal punctuation marks, tables produce dramatic speed improvement, and also reduce the overall space because a large number of states may share a given table.

[0047] Like the character/category byte values, the two low order bits of the tagmark byte value can be used to indicate whether it has an alternative data unit. The bit indicating whether a tagmark is the end of an acceptable string does not matter, since the tagmark itself is never the end of an acceptable string, but always follows a character that is the end of an acceptable string. The first byte value for the tagmark, referred to herein as "CharBase," is 256 - 4N, an integer multiple of 4. Therefore, the two low order bits of each character byte can be masked to obtain six high order bits identifying the character.

[0048] The two low order bits of the table label can similarly be used to provide useful information, if advantageous. For example, they could be used to indicate the length of the pointer to the table.

[0049] The character/category bytes can include an escape code, as described in the mapping application in relation to escape codes. The escape code plus another byte that follows it in the string data can be used to encode characters if the total number of characters is greater than N - 2, which is likely to be the case if N is chosen to optimize the number of pointers of each size. If escape codes are used for all characters that appear at ends of acceptable strings, such as terminal punctuation marks, the terminal punctuation marks may all be collapsed to a single state that also has a single transition back to the start state for a hyphen or other characters that may join the parts of a compound word. But it may be desirable to use an escape code for other characters, so that the choice of when to use escape codes is also a complex tradeoff.

[0050] A wide variety of alternative encodings could be used if advantageous for particular data or for a particular task. For example, the special EOB byte could be omitted if all branches end with a category byte and if every category byte is an end of branch. It might also be possible to use a single bit to indicate whether a character byte or a category byte has an alternative, if a character byte that is final is always followed by a tagmark byte and if a category byte is always final. These approaches could, for example, be used to increase the number of short pointers. The tagmark byte could also be eliminated if distinct blocks of values were allocated to characters and to categories, because the category bytes following a final character byte could be positioned as the first outgoing transitions--presence of a category byte following a character byte would indicate that the character byte is at the end of an acceptable string and that the category byte is at the beginning of a set of category bytes for the string.

[0051] The tagmark byte or its equivalent in other encodings can be thought of as a special transition that follows every final state in an FSM. Following a tagmark transition are a set of bytes that are decoded as categories of strings applicable to all the strings that ended at the final state. Examples of categories include main lexicon, user lexicon, roman numeral, arabic numeral, phone number, date, etc.

[0052] Fig. 7 shows a number of data units within sequence of data units 200. Data units 202, 204, 206, 208, 210, and 212 together form a character subsequence that includes the same information as subsequence 36 in Fig. 1. Data unit 220 is at the beginning of the subsequence's next character subsequence and data unit 222 is at the beginning of the subsequence's alternative character subsequence. Data unit 204, the tagmark byte, is illustratively shown as the next byte of data unit 202, so that the data units of the character subsequence are contiguous, but data unit 220 could instead be the next byte and data unit 204 could be an alternative, in which case the data units of the character would not be contiguous as shown.

[0053] The data units within sequence 200 illustrate all the ranges of byte values in Table I except pointers. Data units 202, 220, and 222 are character bytes. Data unit 204 is a tagmark byte. Data units 206 and 210 are category bytes. And data units 208 and 212 are EOB bytes.

[0054] Each of the character bytes includes two low order bits as shown, the first indicating whether the byte is at the end of an acceptable string and the second indicating whether the byte has an alternative. In the tagmark byte, the first low order bit indicates that the byte is not at the end of an acceptable string and the second indicates whether the byte has an alternative. In each of the category bytes, the first low order bit indicates that the byte is at the end of an acceptable string and the second indicates whether the byte has an alternative.

[0055] The steps in Fig. 8 can replace the steps following box 120 in Fig. 5 if byte values are allocated as in Table I. The step in box 240 increments the location received in box 120 in Fig. 5 and retrieves the byte value at the incremented location. The step in box 242 branches based on whether the retrieved byte value is less than Charbase, meaning that it is a pointer byte value. If so, the step in box 244 goes to the location indicated by the pointer and retrieves the byte at that location.

[0056] When a byte is retrieved that is not a pointer, the step in box 250 subtracts Charbase from the byte's value. If the character received in box 120 has previously been mapped into the range of character byte values by multiplying by four and adding Charbase, it is not necessary to subtract Charbase. The step in box 250 also masks the two low order bits, such as by logically ANDing the byte with the byte 11111100, producing the byte's label.

[0057] The step in box 252 compares the label from box 250 with the character received in box 120. The step in box 254 branches based on the result of the comparison. If the label and character match, the step in box 132 in Fig. 5 is performed, returning the current location.

[0058] If the label and character do not match, the step in box 260 determines whether the label is the table label. If not, there may be further alternatives that should be considered. The step in box 262 obtains the current location's alt bit, which could be saved in the step in box 250. Then the step in box 264 branches based on whether the current location has an alternative. If not, the step in box 142 in Fig. 5 is performed, returning data indicating the character received in box 120 could not be matched.

[0059] If the current location has an alternative, the step in box 266 goes to the alternative's location and retrieves the byte at that location. The alternative can be found by counting alt bits and EOB bytes in the same manner as alts and EOBs are counted in the FSM encoding application and the mapping application. Then the step in box 270 branches based on whether the byte is a pointer, as in box 242. If not, the alternative has been reached so that the step in box 250 can be performed again. If the byte is a pointer, the byte at the location it points to is retrieved as in box 244 before performing the step in box 250.

[0060] When the table label is found in box 260, the step in box 280 uses the character received in box 120 to access the appropriate table entry. As noted above, this can be done by multiplying the character by four and adding the result to the pointer that follows the table label. Then, the test in box 282 branches based on whether the character matches the label byte of the table entry. If the character does not match the label byte, the step in box 142 in Fig. 5 is performed, returning data indicating the character received in box 120 could not be matched. But if the character matches, the location of the table entry is returned in box 282, which has the same effect as returning the current subsequence's location in box 132 in Fig. 5.

[0061] The steps in Fig. 8 can be performed beginning at the location of data unit 202 in Fig. 7. If the next character after the character "x" is a "t," for example, the first iteration of the step in box 254 finds a mismatch with the tagmark label in data unit 204. The step in box 266 counts alt bits and EOB bytes in order to reach data unit 220, and the second iteration of the step in box 254 finds a match with the "t" label, so that the location of data unit 220 is returned in the step in box 132 in Fig. 5.

[0062] Fig. 9 shows steps in obtaining category set data. The steps in Fig. 9 can replace boxes 172, 180, and 184 in Fig. 6 if byte values are allocated as in Table I.

[0063] The step in box 300 retrieves the byte at the location received in box 170 in Fig. 6. The step in box 302 then branches based on whether the byte value is at the end of an acceptable string, as indicated by its final bit. If not, the

current character subsequence is not the ending subsequence of an acceptable string, so no match is returned in box 182. If the byte value is at the end of an acceptable string, the step in box 304 obtains the location of the tagmark byte by calling Advance (Loc, Tagmark). The steps in Figs. 5 and 8 are performed, returning the tagmark's location.

[0064] The step in box 310 goes to and retrieves the next byte of the location returned by Advance. This step is the same as the steps in boxes 240, 242, and 244 in Fig. 8. Then the step in box 312 saves data indicating the six high order bits of the retrieved byte, because those bits indicate one of the categories for the acceptable string. Then, the step in box 320 branches based on the retrieved byte's alt bit. If the alt bit is on, the step in box 322 goes to and retrieves the byte's alternative byte as described above in relation to boxes 266, 270, and 272 in Fig. 8. Then the step in box 312 is performed again. If the alt bit is off, the step in box 324 returns category set data based on the data saved in box 312.

[0065] The steps in Fig. 9 can be performed beginning at the location of data unit 202 in Fig. 7 after matching the character "x." The category set data returned in box 324 indicates the categories "word" and "roman numeral."

[0066] Fig. 10 shows N-bit vector 350 that can be used to save byte values in box 312 and to return category set data in box 324. The step in box 312 can remove the final and alt bits from the byte value and use the higher order bits to indicate which bit in vector 350 should be set to save the byte value. When all the category bytes have been handled in box 312, vector 350 can be returned in box 324 as the category set data. Vector 350 with all zeroes can also be used to return data indicating that a string is not acceptable in box 182 in Fig. 6, since at least one bit must be set if a string is acceptable.

[0067] A more compact data structure can sometimes be obtained by systematically introducing epsilon transitions into an FSM to facilitate sharing of the trailing portions of sets of outgoing transitions. In other words, every state in the FSM can be expanded into a series of binary states, all but the last of which have an epsilon transition to the next binary state in the series. This epsilon-sharing technique also includes minimizing the expanded FSM, after which extra epsilon transitions are removed.

[0068] If the epsilon-sharing technique is used, the order in which the category bytes are arranged following a tagmark byte can be chosen to improve collapsing on the right. For example, the most frequently occurring category can be the last in each set of category bytes, the least frequent first, and so forth. In the allocation of Table I, it may be advantageous to assign the most frequently occurring category to the same byte value as the most frequently occurring character for additional collapsing. In general, the objective is to strengthen the frequency skew to obtain better data compression.

3. Product

[0069] Encoded string data as described above can be used in a recognition system as illustrated in Fig. 11. Fig. 12 shows a software product that can be used in such a system.

[0070] In Fig. 11, processor 370 is connected to receive data defining images from image source 372, which could for example be a scanner. Processor 370 is also connected to provide data indicating recognition results to recognition output 374, which could for example be a display. Processor 370 is also connected to load software from software input device 376, which could be a floppy disk drive, a CD-ROM player, modem, network link, or other peripheral device that can access software on a data storage medium or receive software transmitted through a communication channel.

[0071] In performing recognition, processor 370 accesses instruction memory 380 and executes recognition instructions 382, which include calls to string data access instructions 384. Data memory 390 stores data for access by processor 370. Processor 370 accesses string data 392 in executing string data access instructions 384 and accesses image data 394 in executing recognition instructions 382.

[0072] Software product 400 in Fig. 12 includes data storage medium 402, which could be a floppy disk, a CD-ROM, magnetic tape, or another medium that can store data. Medium 402 stores string data 404 and instructions that can be executed in relation to string data 404.

[0073] In the implementation shown in Fig. 12, the instructions include load instructions 410, starting location instructions 412, advance instructions 414, end status instructions 416, and free instructions 418.

[0074] Load instructions 410 can include a routine that processor 370 executes to load string data 404 into data memory 390. This routine returns the memory location of a header that includes one or more offsets for accessing start states within string data 404. Processor 370 may also load the instructions on medium 402 into instruction memory 380 before execution.

[0075] Starting location instructions 412 can include a routine that processor 370 executes to obtain data indicating the starting location for accessing string data 404 within data memory 390. This routine can be called with the memory location of the header returned by load instructions 410 and with a start state identifier. The routine then accesses the header, obtains the offset to the identified start state, uses the offset to obtain the start state's access location in memory, and returns the access location. For consistency with other instructions, the access location for a start state is the location whose next subsequence is to be matched in response to the initial character of a string.

[0076] Advance instructions 414 can include a routine that processor 370 executes to advance within string data 404, given a location and a character. In other words, the routine can perform steps like those in Fig. 2.

[0077] End status instructions 416 can include a routine that processor 370 executes to obtain data indicating whether a given location is at the end of an acceptable string and, if so, a category set of the string. In other words, the routine can perform steps like those in Fig. 3.

[0078] Free instructions 418 can include a routine that processor 370 executes to flush string data 404 from data memory 390.

[0079] Recognition instructions 382 in Fig. 11 can include a routine that calls the instructions loaded from data storage medium 402. Because of the simplicity of the functions performed by the instructions shown on medium 402, the interface to other software is relatively simple, and software product 400 could be used in a variety of recognition techniques.

[0080] String data 404 can be produced using techniques described in the FSM encoding application and in the spell checking application. The spell checking application describes how an FSM data structure can be obtained.

[0081] In the current implementation, preparation of variations of string data 404 is facilitated by providing a specialized user interface to FSM creation routines. The user can define a category of character strings either with the name of a file that includes a list of strings or with a definition in the form of a special grammar. The grammar can be used, for example, to define a category that has an infinite number of acceptable strings, such as the arabic numbers, arithmetic expressions, or currency amounts; to define a category in which the acceptable strings follow algorithmic rules, such as the roman numbers, phone numbers, or social security numbers; or to extend a category such as a list of strings by adding variations that include punctuation or hyphenation, that combine strings to form compound strings, or that include reasonable patterns of characters similar to those occurring in the strings in the list.

[0082] In the current implementation, a user can provide regular expressions that describe acceptable strings in various ways. For example, an expression can indicate that a string is acceptable if it includes, in sequence, a substring of beginning punctuation, an optional arabic numeral, a substring of separators, an acceptable string from a lexicon, another substring of separators, another acceptable string from the lexicon, and so forth, and ends with a substring of ending punctuation. Or, an expression can indicate that a string is acceptable if it includes, in sequence, a number from 1-9, a substring of from zero to two numbers from 0-9, and then either ends or continues with any number of substrings that include a comma followed by three numbers from 0-9.

[0083] The definition of each category is used to produce a respective FSM. Category data is included in each FSM indicating that each acceptable string of that FSM ends with a tagmark and category identifier. The category for a composite string as described above may be the category of one of its constituent parts, such as the last string.

[0084] If feasible, the FSMs for all the categories can be combined into a single FSM using the well known FSM union algorithm. The combined FSM is determinized and minimized, all without changing the acceptable set of strings. The resulting FSM has one start state. A sequence of states within the FSM may include a cycle, including a pointer from a state that leads back to the same state or to another state that occurred earlier in a sequence of states.

[0085] If it is computationally impractical to combine the FSMs for all categories into a single determinized, minimized FSM, one or more additional FSMs can be included, each with a respective start state and with category data so that each string it accepts ends with a special category identifier.

[0086] The resulting combination of one or more FSMs can then be encoded to produce a data structure like that described in the FSM encoding application. The data structure can be stored on data storage medium 402 in producing software product 400.

C. Applications

[0087] The invention could be applied to recognition problems in many ways, including incremental lookup techniques in which a main recognition routine checks a string by providing each character in sequence, so that if a set of unacceptable candidate strings share an unacceptable prefix, all of them can be eliminated in a single lookup operation. The main recognition routine can also maintain a data structure indicating the locations of previously provided characters so that backtracking to a previous choice point can be more efficient.

[0088] The technique is not limited to recognition of strings of characters, but might also be applied to recognition of strings of other types of elements where the strings fall into discrete categories. For example, in handwriting recognition, strings of strokes fall into word categories. Similarly, in speech recognition, strings of phonemes fall into word categories.

[0089] The technique might also be applied to spelling correction problems. For example, the technique might be applied to sequences of characters from an input device such as a keyboard, to assist in correcting mistakes in typing.

D. Miscellaneous

[0090] The invention has been described in relation to implementations in which a software product is delivered on

a data storage medium. The invention might also be implemented with a software product delivered through transmission, such as through telephone lines or over a network.

[0091] The invention has been described in relation to software implementations, but the invention might be implemented with specialized hardware.

Claims

1. A product (12; 360; 400) for providing information about strings of characters, comprising:

   a data storage medium (390; 402); and
   string data (12; 392; 404) stored by the data storage medium; the string data indicating a set of two or more acceptable strings of characters;
   the string data comprising two or more data units, each of which can be accessed by a processor (14; 370); the data units including, for each of a set of two or more acceptable strings of characters, a respective sequence of data units; the string's sequence (30) of data units beginning with a subsequence of character data units (32, 34; 202, 220, 222) that the processor can access using character data (16) indicating character types of the string's characters; the string's sequence of data units including acceptance data indicating that the string is one of the acceptable strings;

   characterised in that
   the sequence (30) of data units for at least one string in the set of acceptable strings includes a respective ending subsequence (36; 100; 202, 204, 206, 208, 210, 212) of data units that the processor can access after the subsequence of character data units, which ending subsequence includes two or more category data units (206, 210), each category data unit indicating one of a set of two or more categories.

2. The product of claim 1 in which each data unit (202, 204, 206, 208, 210, 212, 220, 222) is a byte.

3. The product of claim 1 or 2 in which the set of acceptable strings includes strings that can be words of a language.

4. The product of any of the preceding claims in which the said at least one string includes an ending having a respective character type; the string's subsequence of character data units including an ending character data unit (202) that includes character label information indicating the ending character's character type; the ending character data unit including a bit indicating that a string at the end of whose respective subsequence of character data units the processor (14; 370) can access the ending character data unit is one of the acceptable strings; the string's acceptance data including the bit.

5. The product of any of the preceding claims in which the string ending data of the said at least one string includes an acceptance data unit (204) in the string's ending subsequence (202, 204, 206, 208, 210, 212); the acceptance data unit having a value indicating that a string at the end of whose respective subsequence of character data units the processor (14; 370) can access the acceptance data unit is one of the acceptable strings; the first string's acceptance data including the acceptance data unit.

6. The product of claim 5 in which the said at least one string's ending subsequence (202, 204, 206, 208, 210, 212) includes a set of category data units (206, 210) that the processor (14; 370) can access after accessing the acceptance data unit (204).

7. The product of any preceding claim in which the set of acceptable strings includes a first string including a respective ending subsequence of data units including two or more category data units and a second string including a respective ending subsequence of data units including one or more category data units, one category dataunit that is also in the first string's ending subsequence being a shared data unit that is also in the second string's ending subsequence.

8. The product of any of the preceding claims in which the string data (12; 392; 404) represent a combined finite state machine (FSM).

9. The product of claim 8 in which one of the set of categories includes infinitely many strings, the words in the set of acceptable words that are in the category being expressible as a cyclic FSM; the combined FSM including the

cyclic FSM.

10. The product of any of the preceding claims, further comprising a processor (14; 370) connected for accessing the string data (12; 392; 404) stored by the data storage medium.

11. A method of operating a system (10; 360) comprising:

a processor (14; 370);
memory (390; 402); the processor being connected for accessing the memory; and
string data (12; 392; 404) stored by the memory; the string data indicating a set of two or more acceptable strings of characters; the string data comprising two or more data units, each of which can be accessed by the processor; the data units including, for each of a set of two or more acceptable strings of characters, a respective sequence of data units; the string's sequence (30) of data units beginning with a subsequence of character data units (32, 34; 202, 220, 222) that the processor can access using character data (16) indicating character types of the string's characters; the string's sequence of data units including string ending data indicating that the string is one of the acceptable strings;
the method comprising:

(A) operating the processor to access the string data using string character data (16) indicating the string's characters; and
(B) operating the processor to obtain category data (18) for the string;

characterised in that
the sequence (30) of data units for at least one string in the set of acceptable strings includes a respective ending subsequence (36; 100; 202, 204, 206, 208, 210, 212) of data units that the processor can access after the subsequence of character data units which ending subsequence includes two or more category data units (206, 210), each category data unit indicating one of a set of two or more categories.

Patentansprüche

1. Ein Produkt (12; 360; 400) zur Lieferung von Informationen über Zeichenketten, das aufweist:

ein Datenspeichermedium (390; 402); und
auf dem Datenspeichermedium gespeicherte Zeichenkettendaten (12; 392; 404); wobei die Zeichenkettendaten einen Satz aus zwei oder mehr akzeptablen Zeichenketten bezeichnen;
wobei die Zeichenkettendaten zwei oder mehr Dateneinheiten aufweisen, auf die ein Prozessor (14; 370) jeweils zugreifen kann; wobei die Dateneinheiten für je einen Satz aus zwei oder mehr akzeptablen Zeichenketten eine entsprechende Sequenz von Dateneinheiten aufweisen; wobei die zeichenkettenbezogene Sequenz (30) aus Dateneinheiten mit einer Untersequenz von Zeichendateneinheiten (32, 34; 202, 220, 222) beginnt, auf die der Prozessor unter Verwendung von Zeichendaten (16) zugreifen kann, die Zeichentypen der Zeichen der Kette bezeichnen; wobei die zeichenkettenbezogene Sequenz aus Dateneinheiten Akzeptanzdaten aufweist, die anzeigen, dass es sich bei der Zeichenkette um eine der akzeptablen Zeichenketten handelt;

dadurch gekennzeichnet, dass
die Sequenz (30) aus Dateneinheiten für mindestens eine Zeichenkette im Satz akzeptabler Zeichenketten eine entsprechende Enduntersequenz (36; 100; 202, 204, 206, 208, 210, 212) aus Dateneinheiten aufweist, auf die der Prozessor nach der Untersequenz von Zeichendateneinheiten zugreifen kann, wobei die Enduntersequenz zwei oder mehr Kategoriedateneinheiten (206, 210) aufweist und jede Kategoriedateneinheit eine aus einem Satz von zwei oder mehr Kategorien bezeichnet.

2. Produkt nach Anspruch 1, wobei jede Dateneinheit (202, 204, 206, 208, 210, 212, 220, 222) ein Byte ist.

3. Produkt nach Anspruch 1 oder 2, wobei der Satz akzeptabler Zeichenketten Zeichenketten aufweist, die Wörter einer Sprache sein können.

4. Produkt nach einem der vorhergehenden Ansprüche, wobei die mindestens eine Zeichenkette eine Endung ein-

schließt, die einen entsprechenden Zeichentyp, aufweist; wobei die Untersequenz aus Zeichendaten der Zeichenkette eine Endzeichendateneinheit (202) aufweist, die Zeichenkennungsinformationen enthält, die den Zeichentyp des Endzeichens bezeichnen; wobei die Endzeichendateneinheit ein Bit einschließt, das anzeigt, dass eine Zeichenkette, bei der ein Prozessor auf die Endzeichendateneinheit am Ende der jeweiligen Untersequenz aus Zeichendateneinheiten der Zeichenkette zugreifen kann, eine der akzeptablen Zeichenketten ist; wobei die Akzeptanzdaten der Zeichenkette dieses Bit einschließen.

5. Produkt nach einem der vorhergehenden Ansprüche, wobei die Zeichenkettenendungsdaten der mindestens einen Zeichenkette eine Akzeptanzdateneinheit (204) in der Enduntersequenz der Zeichenkette (202, 204, 206, 208, 210, 212, 220, 222) aufweist; wobei die Akzeptanzdateneinheit einen Wert aufweist, der anzeigt, dass eine Zeichenkette, bei der der Prozessor (14; 370) auf die Akzeptanzdateneinheit am Ende der jeweiligen Untersequenz aus Zeichendateneinheiten der Zeichenkette zugreifen kann, eine der akzeptablen Zeichenketten ist; wobei die ersten Akzeptanzdaten der Zeichenkette die Akzeptanzdateneinheit einschließen.

6. Produkt nach Anspruch 5, wobei die Enduntersequenz der mindestens einen Zeichenkette (202, 204, 206, 208, 210, 212, 220, 222) einen Satz von Kategoriedateneinheiten (206, 210) aufweist, auf die der Prozessor (14; 370) nach dem Zugriff auf die Akzeptanzdateneinheit (204) zugreifen kann.

7. Produkt nach einem der vorhergehenden Ansprüche, wobei der Satz akzeptabler Zeichenketten eine erste Zeichenkette, die eine entsprechende Enduntersequenz aus Dateneinheiten einschließt, die zwei oder mehr Kategoriedateneinheiten einschließt, und eine zweite Zeichenkette aufweist, die eine entsprechende Enduntersequenz aus Dateneinheiten einschließt, die eine oder mehr Kategoriedateneinheiten einschließt, wobei eine Kategoriedateneinheit, die auch in der Enduntersequenz der ersten Zeichenkette vorhanden ist, eine gemeinsame Dateneinheit darstellt, die auch in der Enduntersequenz der zweiten Zeichenkette vorhanden ist.

8. Produkt nach einem der vorhergehenden Ansprüche, wobei die Zeichenkettendaten (12; 392; 404) eine kombinierte Maschine für den finiten Zustand (FSM) darstellen.

9. Produkt nach Anspruch 8, wobei eine aus dem Satz der Kategorien unendlich viele Zeichenketten aufweist, wobei die Wörter im Satz akzeptabler Wörter, die unter diese Kategorie fallen, als zyklische FSM ausgedrückt werden können; wobei die kombinierte FSM die zyklische FSM einschließt.

10. Produkt nach einem der vorhergehenden Ansprüche, das weiterhin einen Prozessor (14; 370) aufweist, der zum Zweck des Zugriffs auf die Zeichenkettendaten (12; 392; 404) angeschlossen ist, die auf dem Datenspeichermedium gespeichert sind.

11. Verfahren zum Betrieb eines Systems (10; 360), das aufweist:

einen Prozessor (14; 370);
Speicher (390; 402), wobei der Prozessor zwecks Zugriff auf den Speicher angeschlossen ist; und
im Speicher gespeicherte Zeichenkettendaten (12; 392; 404); wobei die Zeichenkettendaten einen Satz aus zwei oder mehr akzeptablen Zeichenketten bezeichnen; wobei die Zeichenkettendaten zwei oder mehr Dateneinheiten aufweisen, auf die der Prozessor jeweils zugreifen kann; wobei die Dateneinheiten für je einen Satz aus zwei oder mehr akzeptablen Zeichenketten eine entsprechende Sequenz von Dateneinheiten aufweisen; wobei die zeichenkettenbezogene Sequenz (30) aus Dateneinheiten mit einer Untersequenz von Zeichendateneinheiten (32, 34; 202, 220, 222) beginnt, auf die der Prozessor unter Verwendung von Zeichendaten (16) zugreifen kann, die Zeichentypen der Zeichen der Kette bezeichnen; wobei die zeichenkettenbezogene Sequenz aus Dateneinheiten Zeichenkettenenddaten aufweist, die anzeigen, dass es sich bei der Zeichenkette um eine der akzeptablen Zeichenketten handelt;
wobei dieses Verfahren aufweist:

(A) den Betrieb des Prozessors zum Zugriff auf die Zeichenkettendaten unter Nutzung von Zeichenkettendaten (16), die die Zeichen der Kette bezeichnen; und
(B) den Betrieb des Prozessors zur Erlangung von Kategoriedaten (18) für die Zeichenkette;

dadurch gekennzeichnet, dass
die Sequenz (30) aus Dateneinheiten für mindestens eine Zeichenkette im Satz akzeptabler Zeichenketten eine entsprechende Enduntersequenz (36; 100; 202, 204, 206, 208, 210, 212) aus Dateneinheiten aufweist, auf die

der Prozessor nach der Untersequenz von Zeichendateneinheiten zugreifen kann, wobei die Enduntersequenz zwei oder mehr Kategoriedateneinheiten (206, 210) aufweist und jede Kategoriedateneinheit eine aus einem Satz von zwei oder mehr Kategorien bezeichnet.

Revendications

1. Produit (12 ; 360 ; 400) de fourniture d'informations concernant des chaînes de caractères, comprenant :

   un moyen de mise en mémoire de données (390 ; 402) ; et

   des données de chaîne (12 ; 392 ; 404) mises en mémoire par le moyen de mise en mémoire de données ; les données de chaînes indiquant un ensemble de deux ou plusieurs chaînes de caractères acceptables ;

   les données de chaînes comprenant deux ou plusieurs unités de données, un processeur (14 ; 370) pouvant accéder à chacune de celles-ci ; les unités de données comprenant, pour chacune faisant partie d'un ensemble de deux ou plusieurs chaînes de caractères acceptables, une séquence respective d'unités de données ; la séquence de la chaîne (30) des unités de données commençant par une sous séquence d'unités de données de caractères (32, 34 ; 202, 220, 222) auxquelles le processeur peut accéder en utilisant des données de caractères (16) qui indiquent les types de caractères des caractères de la chaîne ; la séquence de la chaîne des unités de données comprenant des données d'acceptation indiquant que la chaîne est une des chaînes acceptables ;

   caractérisé en ce que
   la séquence (30) d'unités de données destinée à au moins une chaîne de l'ensemble de chaînes acceptables comprend une sous séquence respective de fin (36 ; 100 ; 202, 204, 206, 208, 210, 212) des unités de données auxquelles le processeur peut accéder après la sous séquence des unités de données de caractère, laquelle sous séquence de fin comprend deux ou plusieurs unités de données de catégorie (206, 210), chaque unité de données de catégorie comprenant l'un des ensembles de deux ou plusieurs catégories.

2. Produit selon la revendication 1, dans lequel chaque unité de données (202, 204, 206, 208, 210, 212, 220, 222) est un multiplet.

3. Produit selon la revendication 1 ou 2, dans lequel l'ensemble de chaînes acceptables comprend des chaînes qui peuvent être les mots d'un langage.

4. Produit selon l'une quelconque des revendications précédantes, dans lequel ladite au moins une chaîne comprend une fin présentant un type de caractère respectif ; la sous séquence de la chaîne d'unités de données de caractère comprenant une unité de données de caractère de fin (202) qui comprend une information d'étiquette de caractère indiquant le type de caractère du caractère de fin ; l'unité de données de caractère de fin comprenant un bit qui indique qu'une chaîne à la fin de laquelle la sous séquence respective d'unités de données de caractère le processeur (14 ; 370) peut accéder à l'unité de données de caractère de fin est l'une des chaînes acceptables ; les données d'acceptation de chaîne comprenant le bit.

5. Produit selon l'une quelconque des revendications précédantes, dans lequel les données de fin de chaîne de ladite au moins une chaîne comprennent une unité de données d'acceptation (204) dans la sous séquence de fin de chaîne (202, 204, 206, 208, 210, 212) ; l'unité de données d'acceptation ayant une valeur indiquant qu'une chaîne à la fin desquelles sous séquences respectives des unités de données de caractère le processeur (14 ; 370) peut accéder à l'unité de données d'acceptation est l'une des chaînes acceptables ; les données d'acceptation de la première chaîne comprenant l'unité de données d'acceptation.

6. Produit selon la revendication 5, dans lequel ladite au moins une sous séquence de fin de chaîne (202, 204, 206, 208, 210, 212) comprend un ensemble d'unités de données de catégorie (206, 210) auxquelles le processeur (14 ; 370) peut accéder après avoir accédé aux unités de données d'acceptation (204).

7. Produit selon l'une quelconque des revendications précédantes, dans lequel l'ensemble de chaînes acceptables comprend une première chaîne comprenant une sous séquence de fin correspondante des unités de données comprenant deux ou plusieurs unités de données de catégorie et une seconde chaîne comprenant une sous

séquence de fin respective des unités de données comprenant une ou plusieurs unités de données de catégorie, une unité de données de catégorie qui se trouve également dans la sous séquence de fin de la première chaîne étant une unité de données partagée qui se trouve également dans la seconde sous séquence de fin de chaîne.

8. Produit selon l'une quelconque des revendications précédantes, dans lequel les données de chaîne (12 ; 392 ; 404) représentent une machine combinée à état fini (FSM).

9. Produit selon la revendication 8, dans lequel l'un des ensembles de catégories comprend des chaînes infiniment nombreuses, les mots qui se trouvent dans l'ensemble des mots acceptables qui sont dans la catégorie pouvant être exprimée par une FSM cyclique ; la FSM combinée comprenant la FSM cyclique.

10. Produit selon l'une quelconque des revendications précédantes, comprenant en outre un processeur (14 ; 370) connecté pour accéder aux données de chaîne (12 ; 392 ; 404) mises en mémoire par le moyen de mise en mémoire de données.

11. Procédé de fonctionnement d'un système (10 ; 360) comprenant :

un processeur (14 ; 370) ;

une mémoire (390 ; 402) ; le processeur étant connecté pour accéder à la mémoire ; et

des données de chaîne (12 ; 392 ; 404) mises en mémoire par la mémoire ; les données de chaîne indiquant un ensemble de deux ou plusieurs chaînes de caractères acceptables ; les données de chaîne comprenant deux ou plusieurs unités de données, à chacune desquelles le processeur peut accéder ; les unités de données comprenant, pour chacune faisant partie d'un ensemble de deux ou plusieurs chaînes de caractères acceptables, une séquence respective d'unités de données ; la séquence de la chaîne (30) des unités de données commençant par une sous séquence d'unités de données de caractère (32, 34 ; 202, 220, 222) auxquelles le processeur peut accéder en utilisant les données de caractère (16) qui indiquent les types de caractère des caractères de la chaîne ; la séquence de la chaîne des unités de données comprenant des données de fin de chaîne indiquant que la chaîne est l'une des chaînes acceptables ;

le procédé comprenant :

(A) la mise en action du processeur pour accéder aux données de chaîne en utilisant les données de caractère de chaîne (16) qui indiquent les caractères de chaîne ; et

(B) la mise en action du processeur pour obtenir les données de catégorie (18) destinées à la chaîne ;

caractérisé en ce que

la séquence (30) des unités de données destinée à au moins une chaîne dans l'ensemble des chaînes acceptables comprend une sous séquence respective de fin (36 ; 100 ; 202, 204, 206, 208, 210, 212) des unités de données auxquelles le processeur peut accéder après la sous séquence des unités de données de caractère, laquelle sous séquence de fin comprend deux ou plusieurs unités de données de catégorie (206, 210), chaque unité de données de catégorie indiquant l'un des ensembles de deux ou plusieurs catégories.

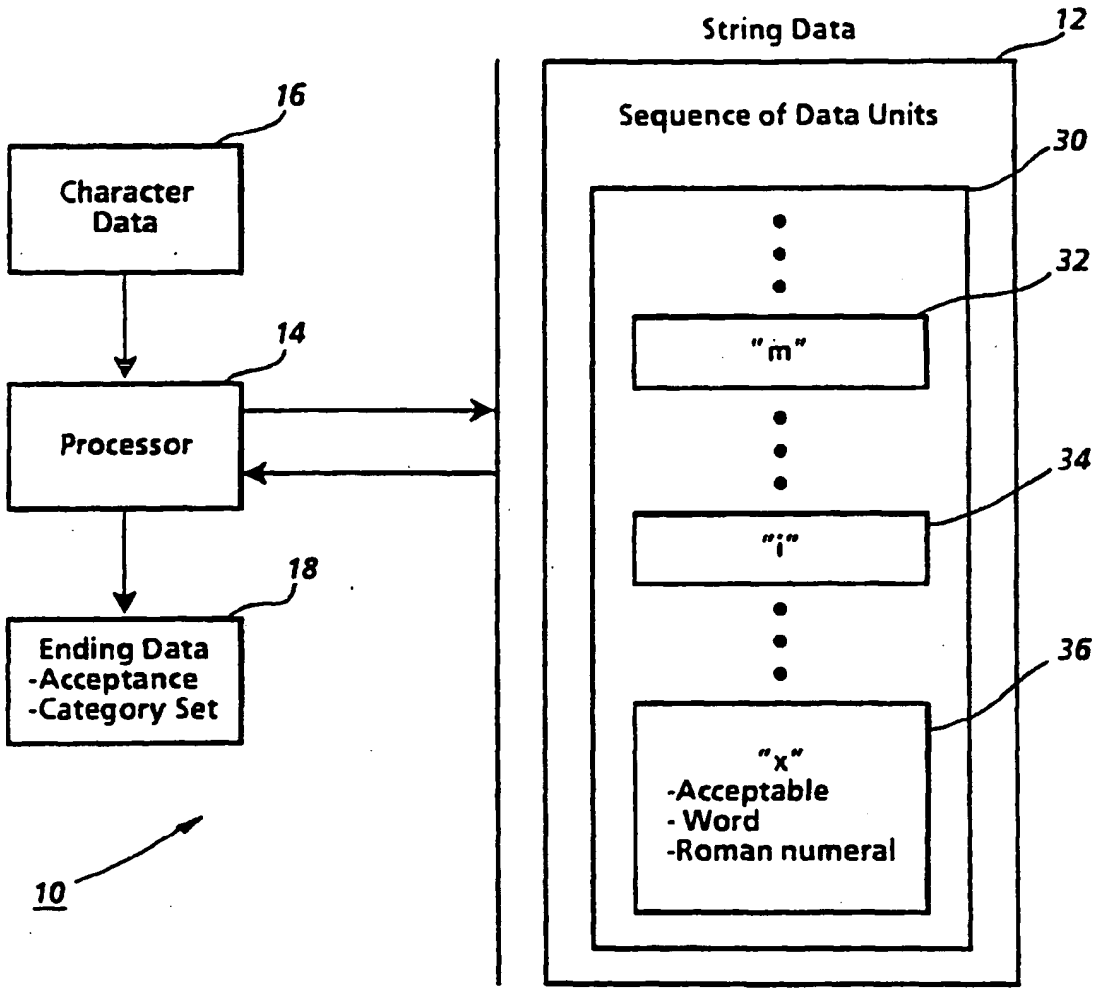

String Data
12
Sequence of Data Units
30
16
Character Data
14
Processor
18
Ending Data
-Acceptance
-Category Set
32
"m"
34
"i"
36
"x"
-Acceptable
- Word
-Roman numeral
10

Fig. 1

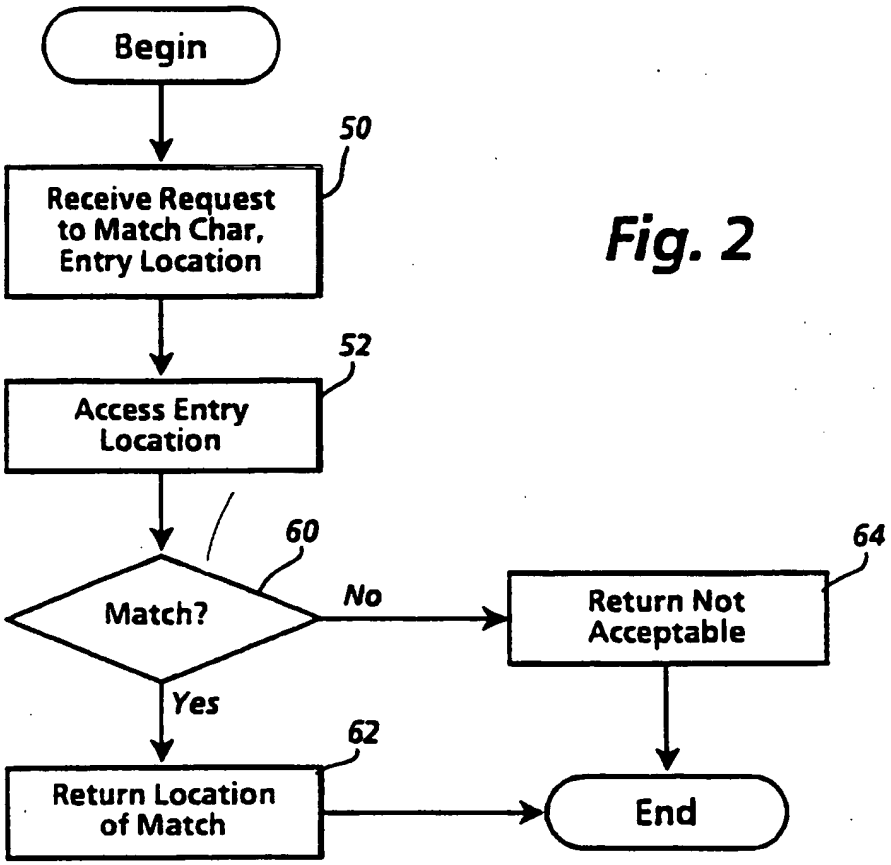
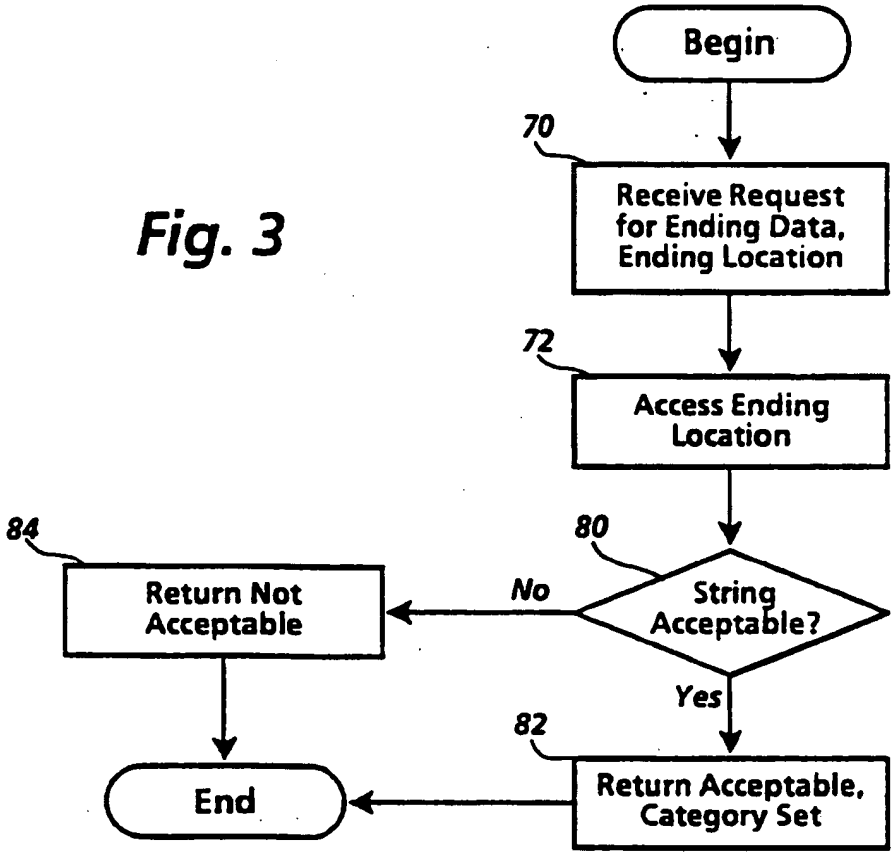

Begin
50
Receive Request to Match Char, Entry Location
52
Access Entry Location
60
Match?
No
64
Return Not Acceptable
Yes
62
Return Location of Match
End

Fig. 2

Fig. 3


Begin
70
Receive Request for Ending Data, Ending Location
72
Access Ending Location
80
String Acceptable?
No
84
Return Not Acceptable
End
Yes
82
Return Acceptable, Category Set

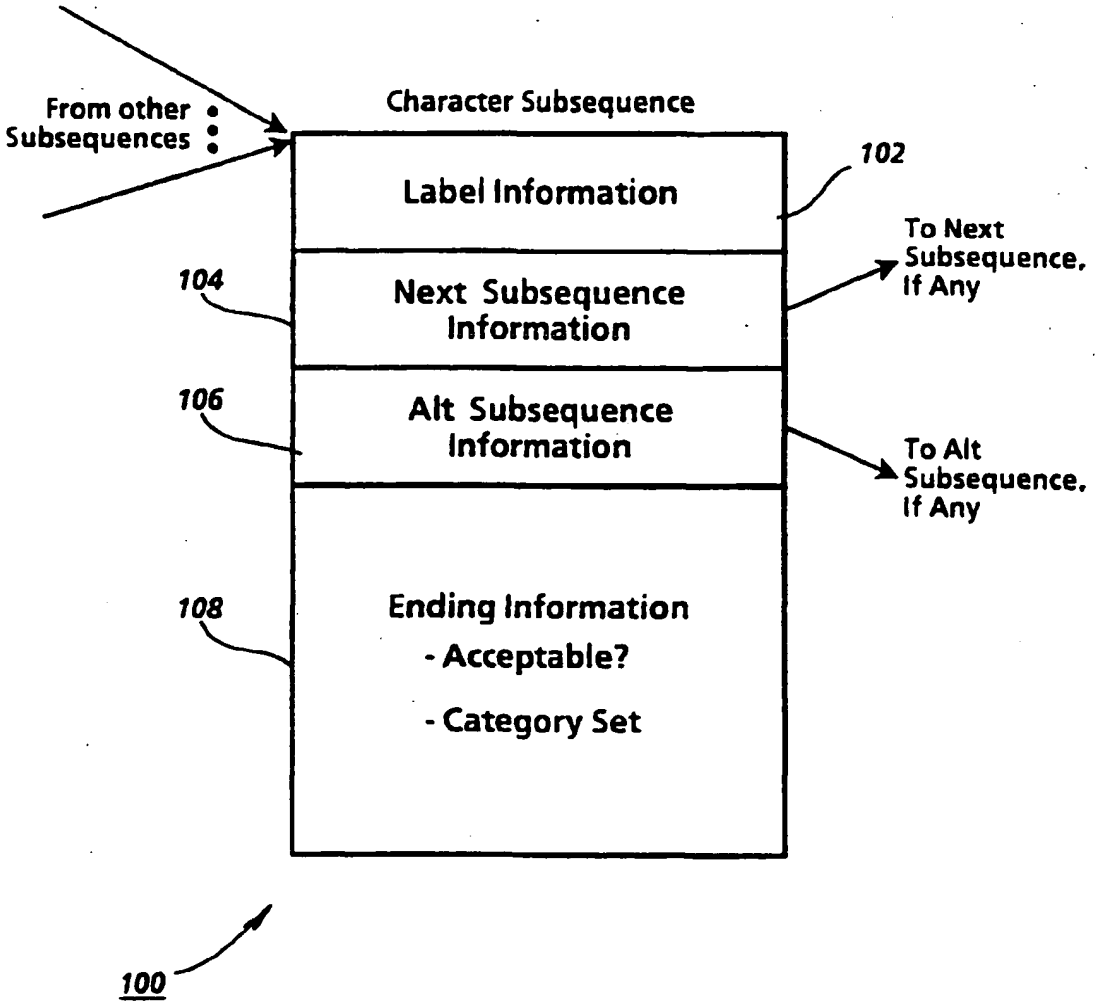

From other Subsequences
Character Subsequence
Label Information
102
To Next Subsequence, If Any
104
Next Subsequence Information
106
Alt Subsequence Information
To Alt Subsequence, If Any
108
Ending Information
- Acceptable?
- Category Set
100

Fig. 4

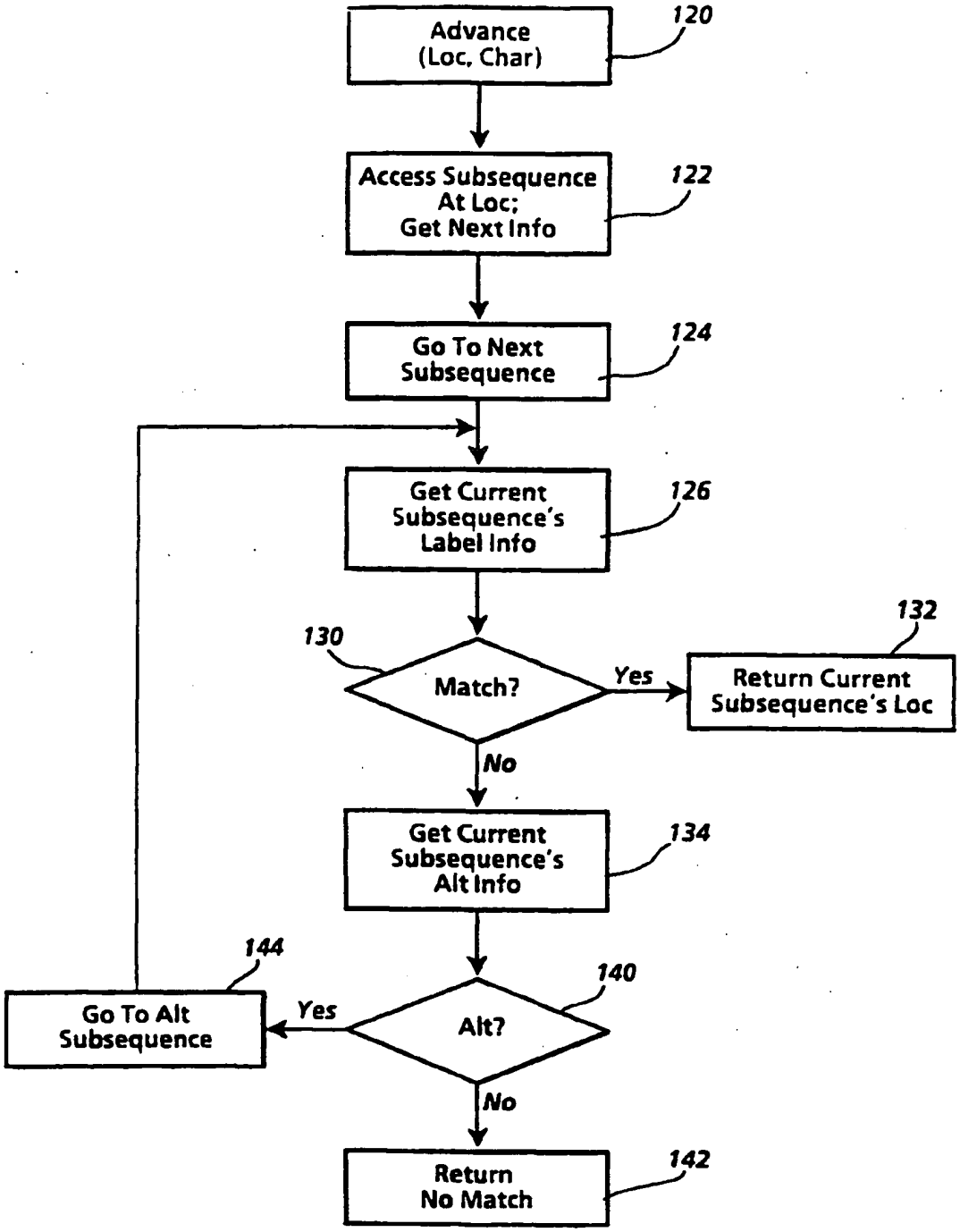

Advance
(Loc, Char)
120
Access Subsequence
At Loc;
Get Next Info
122
Go To Next
Subsequence
124
Get Current
Subsequence's
Label Info
126
130
Match?
Yes
132
Return Current
Subsequence's Loc
No
Get Current
Subsequence's
Alt Info
134
144
Go To Alt
Subsequence
Yes
140
Alt?
No
Return
No Match
142

Fig. 5

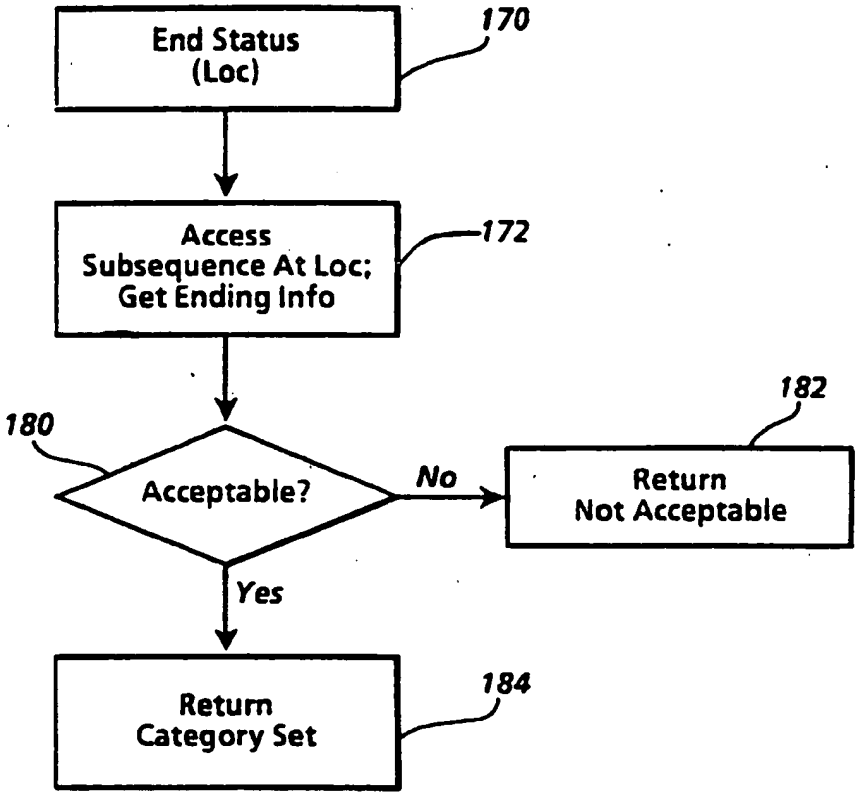

End Status
(Loc)
170
Access
Subsequence At Loc;
Get Ending Info
172
180
Acceptable?
No
182
Return
Not Acceptable
Yes
Return
Category Set
184

Fig. 6

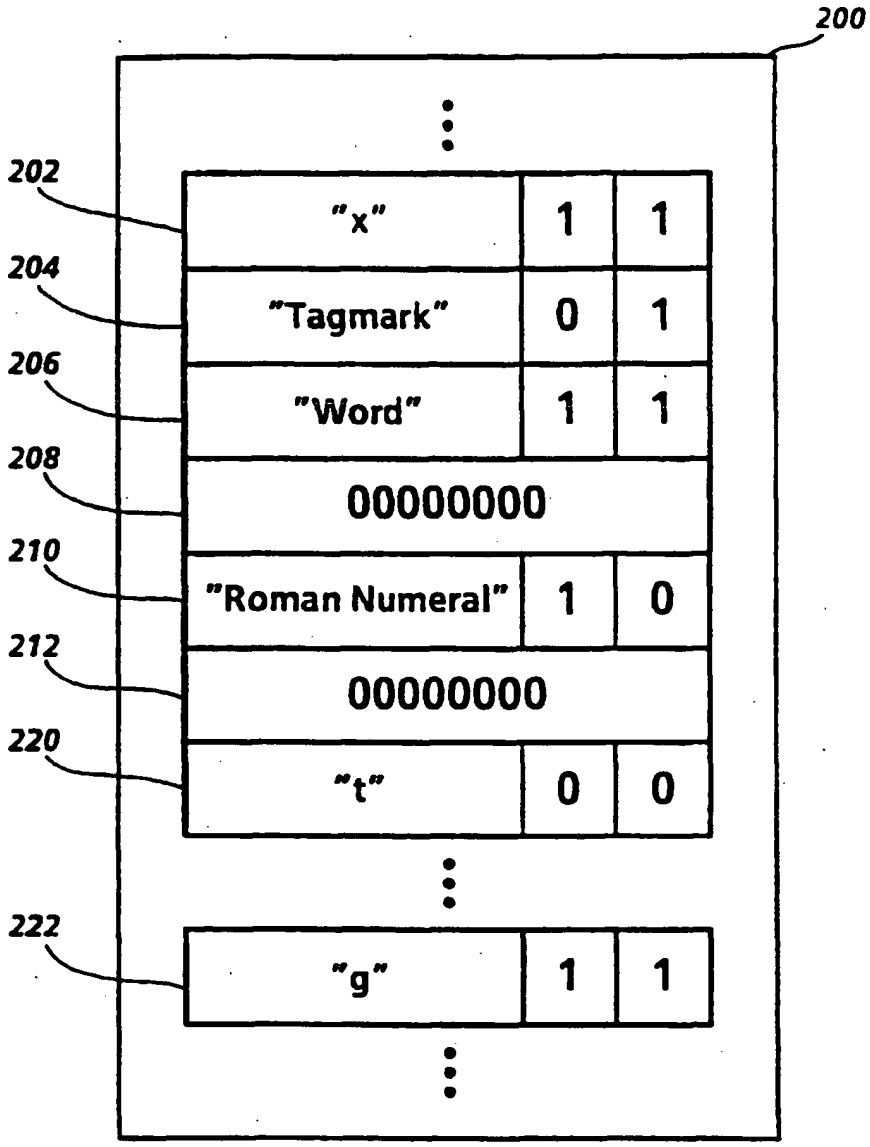

200
202
"x"
1
1
204
"Tagmark"
0
1
206
"Word"
1
1
208
00000000
210
"Roman Numeral"
1
0
212
00000000
220
"t"
0
0
222
"g"
1
1

Fig. 7

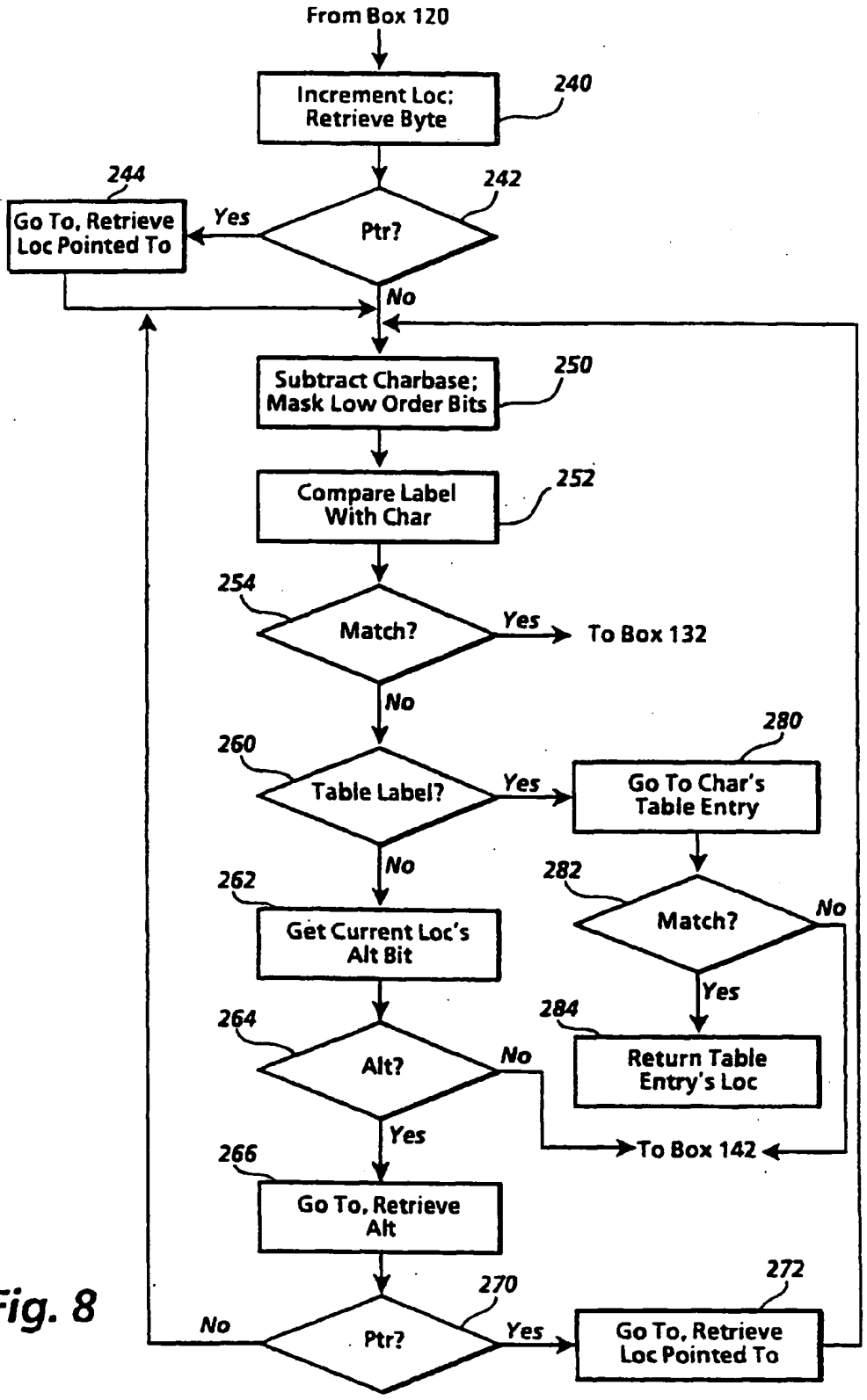

From Box 120
Increment Loc; Retrieve Byte
240
Ptr?
242
Yes
Go To, Retrieve Loc Pointed To
244
No
Subtract Charbase; Mask Low Order Bits
250
Compare Label With Char
252
Match?
254
Yes
To Box 132
No
Table Label?
260
Yes
Go To Char's Table Entry
280
Match?
282
No
Yes
Get Current Loc's Alt Bit
262
Return Table Entry's Loc
284
Alt?
264
No
Yes
To Box 142
Go To, Retrieve Alt
266
Ptr?
270
No
Yes
Go To, Retrieve Loc Pointed To
272

Fig. 8

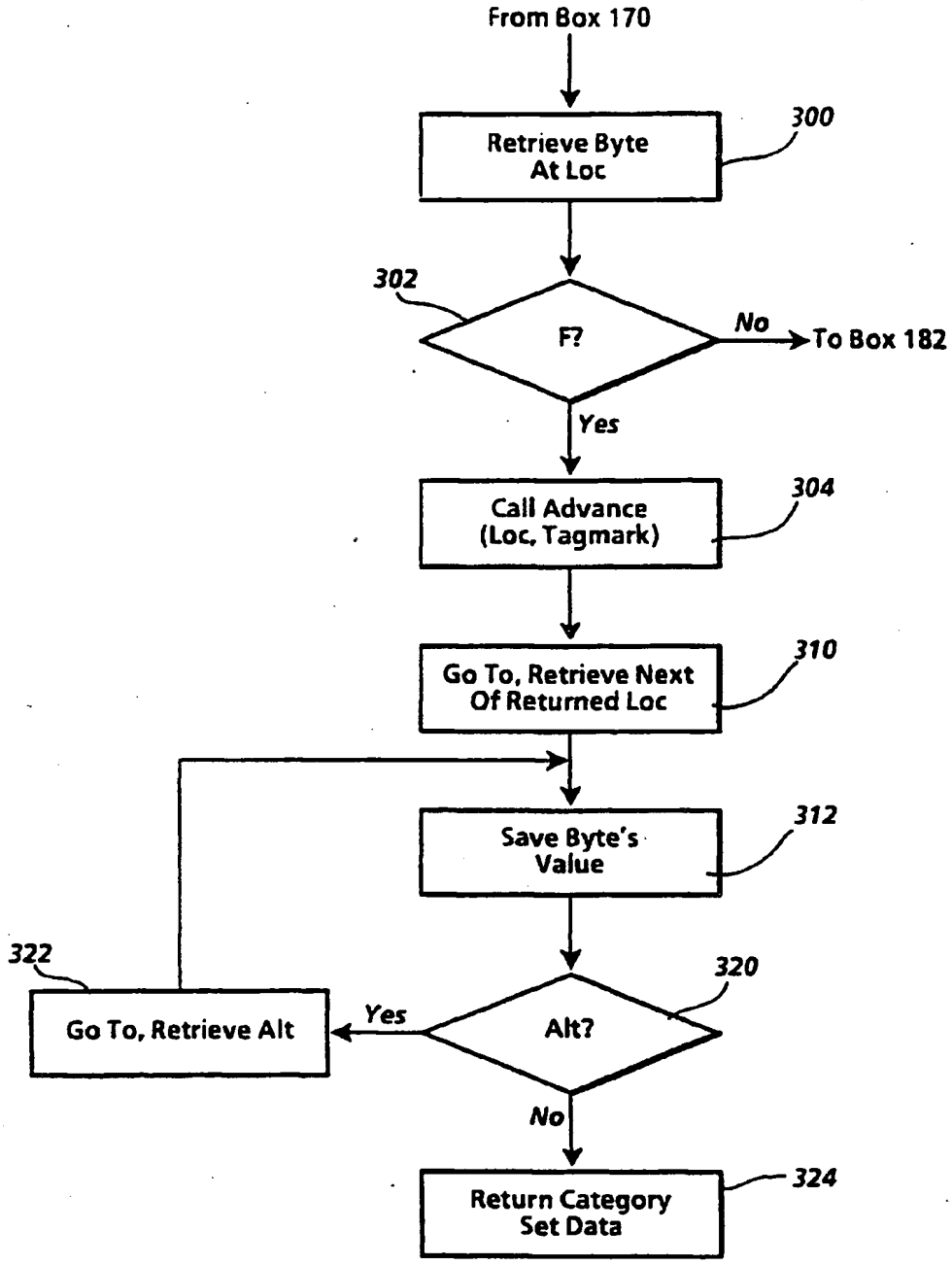

From Box 170
Retrieve Byte At Loc
300
302
F?
No
To Box 182
Yes
Call Advance (Loc, Tagmark)
304
Go To, Retrieve Next Of Returned Loc
310
Save Byte's Value
312
322
Go To, Retrieve Alt
Yes
320
Alt?
No
Return Category Set Data
324

Fig. 9

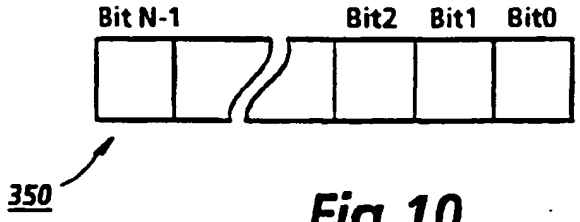

Bit N-1
Bit2
Bit1
Bit0
350

Fig.10

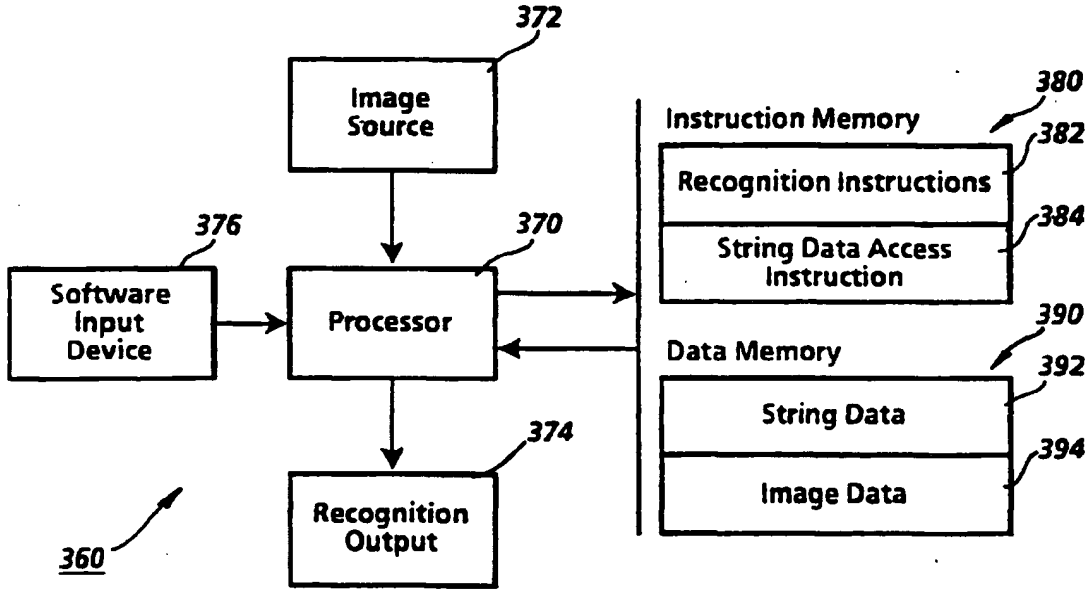

372
Image Source
380
Instruction Memory
382
Recognition Instructions
384
String Data Access Instruction
376
Software Input Device
370
Processor
390
Data Memory
392
String Data
394
Image Data
374
Recognition Output
360

Fig.11

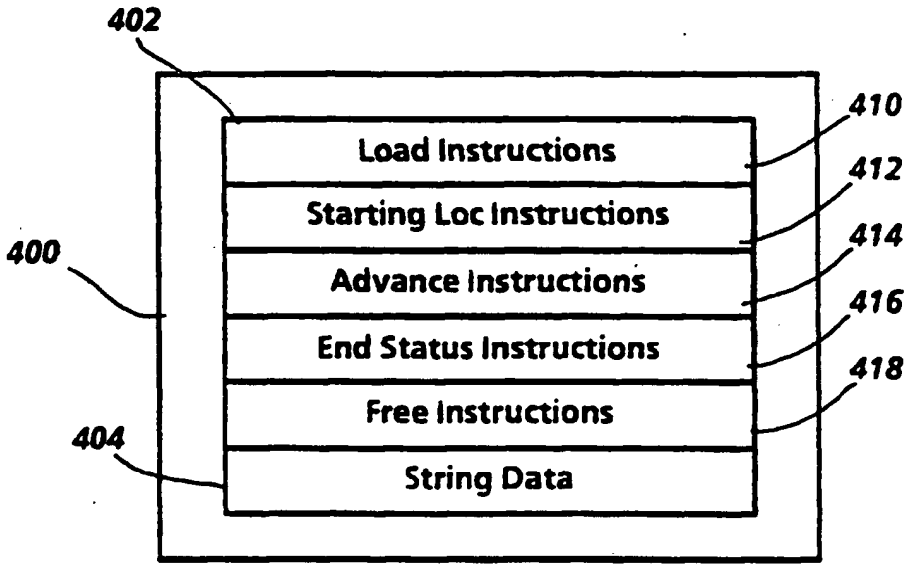

402
410
Load Instructions
412
Starting Loc Instructions
400
414
Advance Instructions
416
End Status Instructions
418
Free Instructions
404
String Data

Fig.12